Assignment 2

Derek Paulsen

1 Problem 1

1.1 Part 1

We can update the flow by finding a path in the residual graph from s to u and then finding a path from v to t, when then push an extra unit of flow through this path if it exists. Find the path from s to u and from v to t can be done using breadth first search each requiring $O(m)$ time, meaning that this is operation overall takes $O(m)$ time.

1.2 Part 2

Define a potential $p(u) =$ cheapest path from s to u before increasing the cost of (u, v), because the initial flow is optimal, this potential will result in $\epsilon \geq 0$. Additionally, since the costs are all integral, we know that all reduced costs are multiples of 1 (i.e. they are integral). Because all reduced costs are a multiple of the increment, in any edge with reduced cost $c_p(x, y) > 0$ included in a cycle will make the cycle's cost non-negative. Since the previous flow was optimal we only need to consider cycles that decrease the flow over the edge which has it's cost incremented. This means that we only consider paths from u to v where all the edges have $c_p(x, y) = 0$. Now run max flow from u to v using only edges with $c_p(x, y) = 0$.

Runtime

p can be computed with Bellman Ford in $O(mn)$ time, finding the edges with $c_p(u, v) = 0$ is $O(m)$, running max flow is $O(mn)$, therefore the whole procedure runs in $O(mn)$ time.

Algorithm

Below is our algorithm, note that we omit the capacities for ease of exposition, however the maxflow algorithm step runs with respect to the capacities of the original graph.

- IncrementCost$(V, E, f, \hat{c}, (u, v), inc)$
- compute $p(w)$ = the cheapest path from s to w
- $E' \leftarrow \{(w, z) | (w, z) \in E, \hat{c}(w, z) + p(w) - p(z) = 0\}$
- $\hat{c}(u, v) \leftarrow \hat{c}(u, v) + inc$ // increment the cost the edge
- Run maxflow from u to v over the graph (V, E') with starting flow f to get f'
- Return $f', \hat{c}$

1.3 Part 3

Let $c(u, v)$ be the true cost of an edge and $\hat{c}(u, v)$ be the current cost of the edge in the graph during algorithm execution. Our algorithm works as follows. We begin by running maxflow over the input graph and then setting the cost of every edge to be 0. We then iteratively increment the costs on the edges by decreasing powers of two mantaining the invariant that the costs of all edges are always a multiple of the cost increment. Because the increment is always a multiple of the current costs of the edges we can use the algorithm from part 2 to correctly update the flow in $O(mn)$ time. The algorithm the terminates once our increment becomes < 1 at which point $\forall(u, v) \in E, c(u, v) = \hat{c}(u, v)$ since all of our costs are integral. Because at every step the flow is optimal relative to $\hat{c}$, $c(u, v) = \hat{c}(u, v)$ implies that the cost is optimal to c, hence we terminate with a min cost maximum flow for the original input graph.

- CostScaling(V, E, c)
- $\beta \leftarrow \max_{(u,v)\in E}(\lfloor \log_2(c(u,v)) \rfloor)$ // the largest power of two that is less than or equal to the maximum cost
- $\hat{c}(u,v) = 0$ // all edge costs are zero to start
- $f \leftarrow$ maxflow from s to t over (V, E)
- While $\beta \geq 0$
  - For each $(u,v) \in E$ where $\hat{c}(u,v) + 2^\beta \leq c(u,v)$ // each edge that can be incremented with the current increment
    - $f, \hat{c} \leftarrow$ IncrementCost$(V, E, f, \hat{c}, (u,v), 2^\beta)$
  - $\beta \leftarrow \beta - 1$ //halve increment

Runtime

The outerloop runs $O(\log_2(C))$ and the inner loop in the worse case makes m calls to the procedure from part 2. Hence we make $O(m \log C)$ calls to the procedure from part 2. This results in an overall runtime of $O(m^2 n \log C)$.

2 Problem 2

2.1 Part 1

We start with a high level overview of our algorithm. We begin by running a max flow algorithm over the graph G. We then check if the flow is feasible (i.e. satisfies all of the lower bounds for each edge). If the flow is not feasible then we find cycles in the resdiual graph and push flow around the cycles in the satisfy the lower bounds of each edge. We call this initial flow feasible flow f. We then update the capacity on each edge to be $f(e) - l(e)$ (the amount of slack that we have in that edge) and reverse all edges in the graph to get a new graph G'. Then run maxflow from t to s on the updated graph to get a new maxflow f'. Finally substract this flow from the original flow to get $f - f' = f^*$ which is the final minflow.
First we define a $\gamma((u,v), f)$ to be the maximum amount of flow that can be passed through an edge in the residual graph H, such that the resulting flow doesn't violate the constraints. Formally, this is

$$\gamma((u,v), f) = \begin{cases} u(e) - f(e) \text{ if } (u,v) \in E \text{ (a forward edge in the graph)} \\ f(e) - l(e) \text{ if } (v,u) \in E \text{ (a backward edge in the graph)} \end{cases}$$

We then define $\Gamma(C, f)$ where C is a cycle in the residual graph H with flow f to be

$$\Gamma(C, f) = \min_{(u,v)\in C} \{\gamma((u,v), f)\}$$

- Run any maxflow algorithm over G with capacities = u to get flow f
- While f is not feasible (there are edges which violate the lower bounds)
  - $e \leftarrow$ an edge which has $f(e) < l(e)$
  - find an cycle C such that $e \in C, \Gamma(C, f) > 0$
  - push $\Gamma(C, f)$ units of flow around C and update f
- Update the capacity of each edge $e \in E$ to be $f(e) - l(e)$ and reverse all of the edges in the graph, to get a new graph G'
- run max flow from t to s on G' to obtain f'
- $f^* \leftarrow f - f'$
- return f^*

2.2 Part 2

Let P be the set of all $s-t$ paths and cycles which have no repeating edges. Let x_p be the flow running through path or cycle $p \in P$. Let

$$A \in \mathbb{Z}^{|E| \times |P|} = A_{e,p} = \begin{cases} 1 \text{ if } e \in p \\ 0 \text{ otherwise} \end{cases}$$

Consider the following LP,

$$\begin{aligned} \min_x \quad & \sum_{p \in P} x_p \\ \text{subject to} \\ & Ax \leq u \\ & Ax \geq l \\ & x \geq 0 \end{aligned}$$

This is the min flow formulated as an LP. We can then rearrgage,

$$\begin{aligned} \min_x \quad & \sum_{p \in P} x_p \\ \text{subject to} \\ & -Ax \geq -u \\ & Ax \geq l \\ & x \geq 0 \end{aligned}$$

and take the dual,

$$\begin{aligned} \max_{y,z} \quad & y^T l - z^T u \\ \text{subject to} \\ & y^T A - y_i^T A \leq 1 \\ & y, y_i \geq 0 \end{aligned}$$

We now claim that y, z define the forward and backward edges in the max $s-t$ cut of the graph G.

Lemma : y, z are integer

We first note that the solution to the dual is integer because all entries of A are 0 or 1, hence for any basis B of A, $det(B) \in \{-1, 0, 1\}$. By Cramer's rule, any basic feasible solution to the dual will have integer entries.

Lemma : y, z are disjoint and 0,1 for some optimal solution

Next we note that $y^T A - z^T A \leq 1 \implies y - z \leq 1$. Taking a look at the objective of the dual we note that $u \geq l, \forall i = 1, ..., |P|, h > 0, l_i(h+1) - u_i h \leq l_i$ this implies there exists an optimal solution where $y, z \leq 1$ and they are disjoint because for any feasible solution where $\exists i, yi > 1, z_i > 0$, we could obtain a new feasible solution with objective value at least as good by setting $y_i = y_i - z_i$ and $z_i = 0$.

Meaning of the constriants

A path $p \in P$. For any $s-t$ cut of G, a path must cross the cut an odd number of times. $y^T A - y_i^T A \leq 1$ simply means that the cut must cut the backward and forward edges. This is because if a path p crosses a cut $(S, V \setminus S)$, $2k+1$ times, then there must be $k+1$ forward edges and k backward edges.

Conclusion

Finally, we observe that the objective of the dual is equal to the capacity of the maximum cut, by strong duality we have that if the primal has an optimal solution so does the dual and their respective objective values are equal. Therefore we have that the min flow is equal to the max cut.

2.3 Part 3

We reduce the problem to a min flow problem by defining a graph G. We begin by defining our vertexes

$$\begin{aligned} V &= \{s, t\} \text{ (source and sink)} \\ &\cup \{u_i | i = 1, ...n\} \text{(start of class)} \\ &\cup \{v_i | i = 1, ...n\} \text{(end of class)} \end{aligned}$$

Next we define the edges,

$$\begin{aligned} E = &\{(s, u_i) | i = 1, ...n\} \text{(source to start of each class, new student attends a class)} \\ &\cup \{(v_i, t) | i = 1, ...n\} \text{(end of each class to the sink, student goes home)} \\ &\cup \{(u_i, v_i) | i = 1, ...n\} \text{(one edge per class, class attendance)} \\ &\cup \{(v_i, u_j) | i, j = 1, ...n, b_i + r_{i,j} \leq a_j\} \text{(an edge from the end of a class to another if there is time to get there)} \end{aligned}$$

We then define the lower bounds to be

$$l : E \to \mathbb{Z}^+ \quad l(e) = \begin{cases} 1 \text{ if } e = (u_i, v_i) \\ 0 \text{ otherwise} \end{cases}$$

finally we define all upper bounds to be n,

$$u : E \to \mathbb{Z}^+ \quad u(e) = n$$

Each unit of flow to this problem corresponds to a single student, if that unit of flow crosses edges (u_i, v_i) then that student attends lecture i. The student can then either attend another lecture if time permits corresponding to following some edge in $\{(v_i, u_j) | i, j = 1, ...n, b_i + r_{i,j} \leq a_j\}$ or can go home (following the edge of the end of the lecture to the sink, t). Since the lower bound on all the edges corresponding to lectures have lower bound 1 any valid solution to this min flow problem corresponds to a valid assignment of students to lectures. Finally, we minimize the flow which corresponds to the minimizing the number of students required to cover all lectures.

3 Problem 3

3.1 Part 1

Suppose we construct a bipartite graph $G = \{E, U, V\}$ where for each student $u \in U$ there is an edge (u, v) if the student wishes to meet with faculty $v \in V$. By Konig's Theorm we have that the maximum matching in this graph is equal in size to the minimum vertex cover. We note that $|E| = d|V|$ and that each vertex in the graph covers at most d edges, therefore the size of the minimum vertex cover is at least $|V|$ therefore the maximum matching is at least $|V|$, hence there exists a perfect matching in the graph. A perfect matching corresponds to assigning each student to meet with a single faculty member, this concludes the proof.

3.2 Part 2

Suppose we do a single round of matching where each student meets with a single faculty member. At the end of this round we remove the edges that were used in the matching. This decreases the number of edges by $|V|$ meaning that the degree of each student and faculty decreases by 1. However now we can apply the same proof as above, there are $(d-1)|V|$ edges in the graph and each vertex covers at most $d-1$ edges therefore the minimum vertex cover is still of size $|V|$ hence there exists a perfect matching. After d rounds of matching we have removed $d|V| = |E|$ edges meaning that all students have meet with all the faculty that they wanted to.

3.3 Part 3

Consider adding 'fake' edges such that each student has out degree s and each professor has in degree s. From part 2 we know that we can complete all assignments in s steps. At each step if a 'fake' edge is choosen then the time slot for the corresponding professor and student is free for that round. This produces a schedule where all desired meetings take place is s time slots, which completes the proof.

4 Problem 4

For this we first define our min cost flow problem.
Let $\delta_{in}(v)$ be the number of edges entering vertex v and $\delta_{out}(v)$ be the number of edges leaving vertex v. Let $G = \{V, E\}$ be the input graph. First construct two sets $X = \{v \in V | \delta_{in}(v) > \delta_{out}(v)\}$ and $Y = \{v \in V | \delta_{in}(v) < \delta_{out}(v)\}$.
Now we construct a graph, $G' = \{E', X, Y, s, t\}$ and define a cost function $c : E' \to \mathbb{R}$ and a capacity function $u : E' \to \mathbb{R}$.
We define $E' = X \times Y \cup \{(s, x) | x \in X\} \cup \{(y, t) | y \in Y\}$. We then define our capacity function,

$$u(x, y) = \begin{cases} \delta_{in}(y) - \delta_{out}(y) \text{ if } x = s \\ \delta_{out}(x) - \delta_{in}(x) \text{ if } y = t \\ \delta_{in}(x) - \delta_{out}(x) \text{ else} \end{cases}$$

We then define our cost function,

$$c(x, y) = \begin{cases} 0 \text{ if } x = s \vee y = t \\ \text{cheapest path from } (x, y) \text{ in } G \text{ else} \end{cases}$$

We then run any min cost max flow algorithm over this graph. The resulting min cost flow f^* is then used to modify G by adding $f^*(x, y)$ 'fake' edges between x, y in the the graph G. We then use a standard algorithm for finding the Eulerian path in the modified graph G to give us a path P. We then modify this path by replacing any 'fake' edge (x, y) that we added in the last step with the shortest path between the x and y (which we computed during the construction of G').

Optimality

Suppose that for our input graph we must add at least n edges to the graph to create a graph with an Eulerian path. Now suppose that we have a different modification to the graph which adds $k > 0, n + 2k$ edges to the graph while still allow for an Eulerian graph. By construction we there is some way to remove $2k$ edges to get a strictly better path, hence any optimal solution will only add the minimum number of edges required to make the graph Eulerian.
Now consider our solution, we note that we add the minimum number of edges to the graph to make it Eulerian. By construction we add these 'fake' edges in the cheapest possible way, hence our solution produces an optimal Eulerian path through the graph.